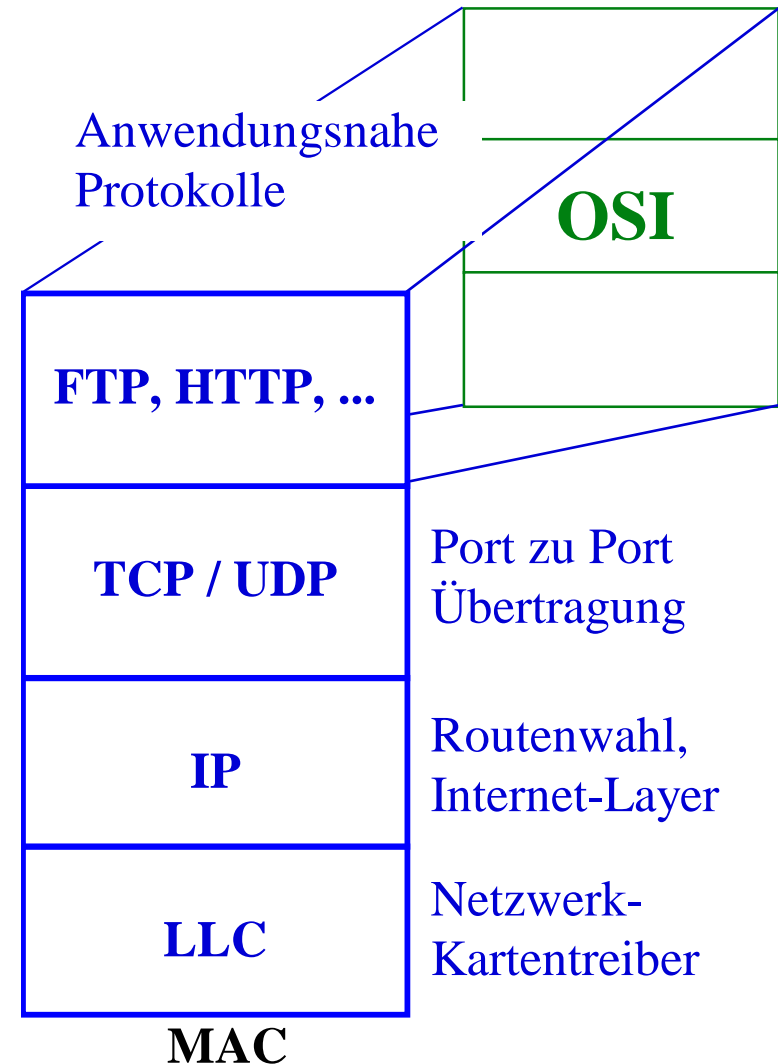


12. Kommunikation im Internet

12.1. Internet-Protokollhierarchie

- Historisch gewachsen in der Unix-Kultur.
- Vier Schichten anstelle von 7 in OSI.
- Die oberste Schicht umfasst:
 - OSI Application Layer,
 - OSI Presentation Layer,
 - OSI Session Layer.
- 4 Internet Schichten:
 - High-Level Protokolle,
 - Transport Protokolle,
 - Netzwerk Protokolle,
 - Logical Link Control.
- Media Access Control:
 - Unterhalb von LLC sind viele verschiedene WAN, LAN Techniken angesiedelt.



12.1.1 TCP/IP Protokollfamilie

- UDP:

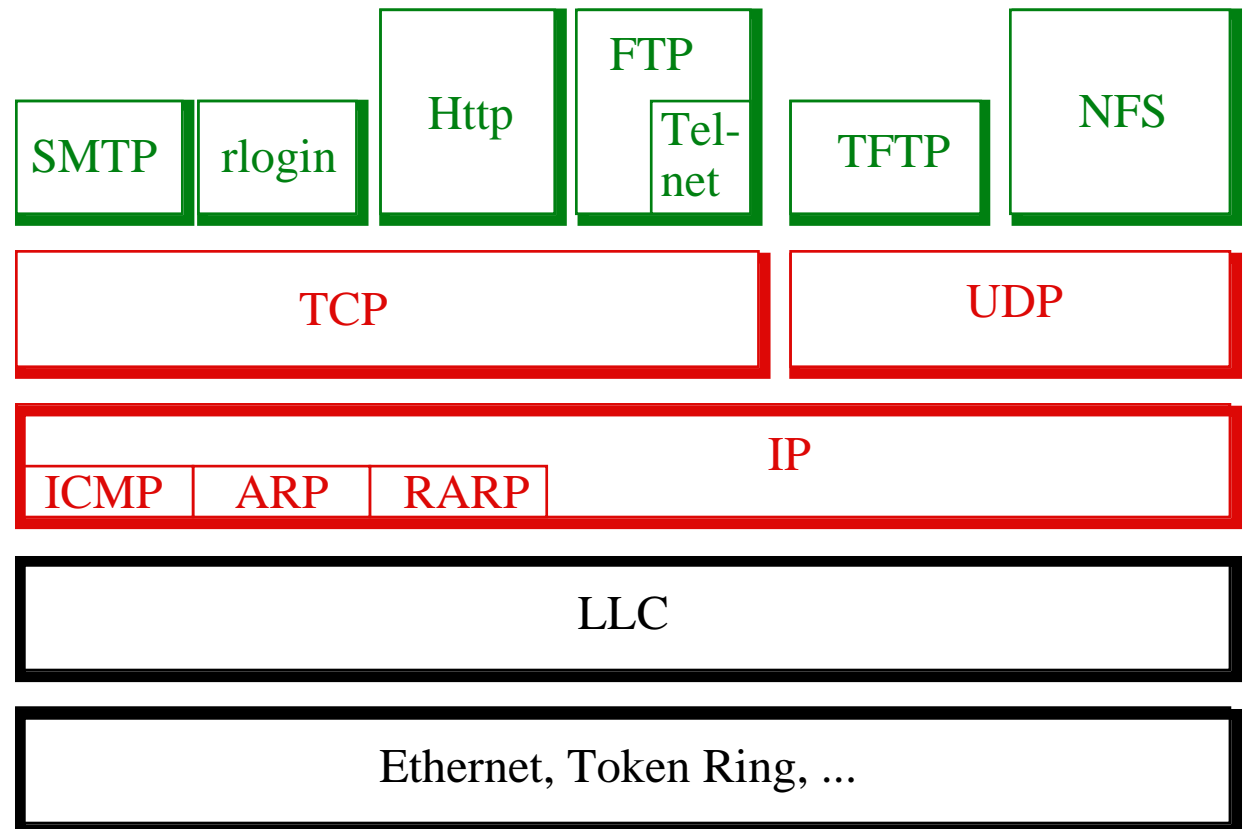
- Verbindungsaufbau entfällt,
- keine Ablieferungsgarantie,
- keine Sequenzgarantie.

- TCP:

- Verbindungsaufbauphase,
- Sichere Ablieferung,
- Fluss-Steuerung ...

- IP Subprotokolle:

- zur Adressfindung,
- Allg. Steuerung,
- Routing ...



- Java-Klassen stützen sich übrigens ausschliesslich auf IP.

12.1.2 TCP/IP Paket:

- Hier TCP / IP Kommunikation über ein Ethernet.

- **LLC Schicht:**

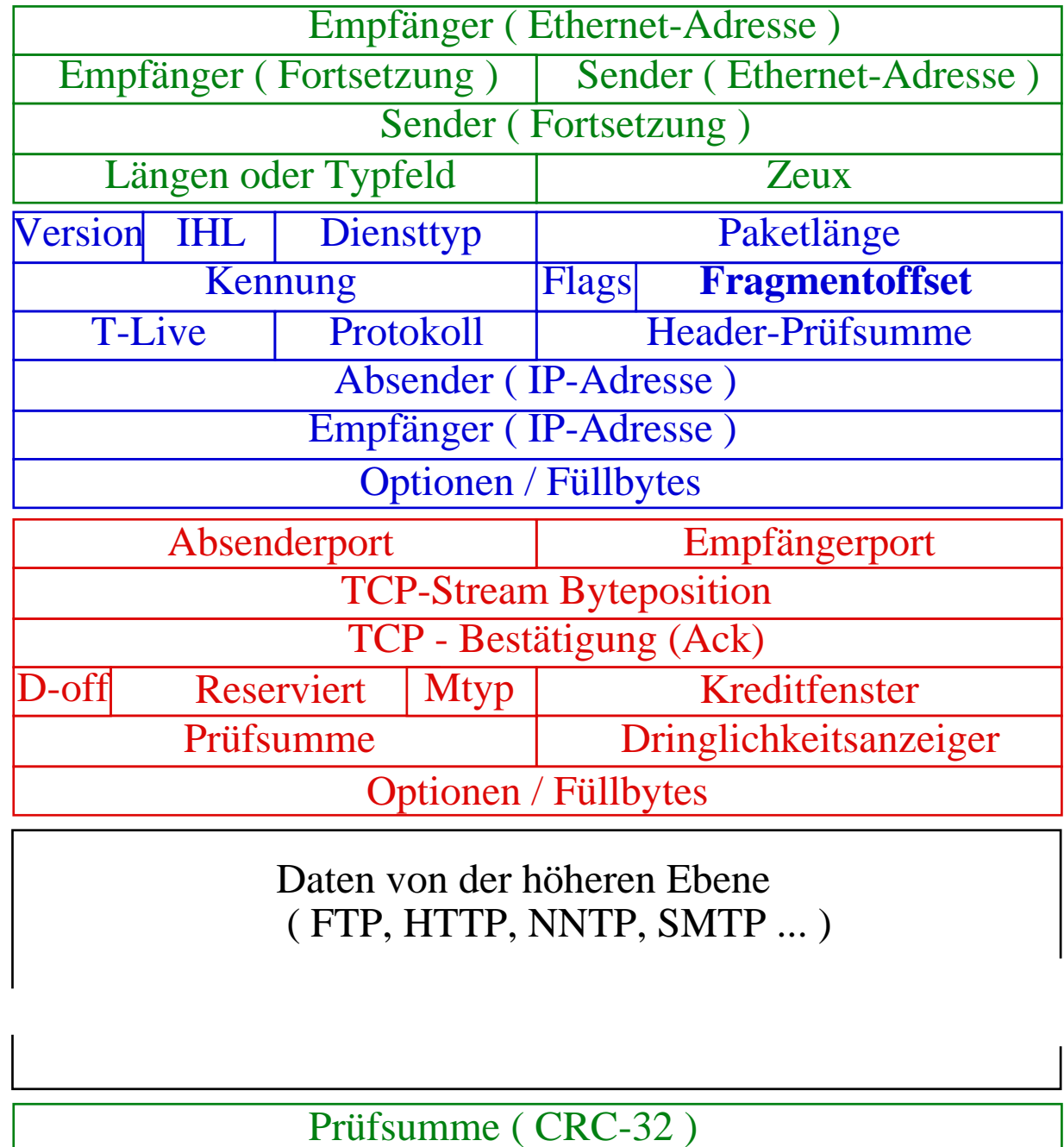
- MAC-Adressen,
- Typ/Längenfeld,
- Evtl. SNAP ...

- **IP Schicht:**

- IP-Adressen,
- QoS/Diensttyp
- Nächster Pakethandler,
- Position im 8 Bytestrom,
- Hop-Count ...

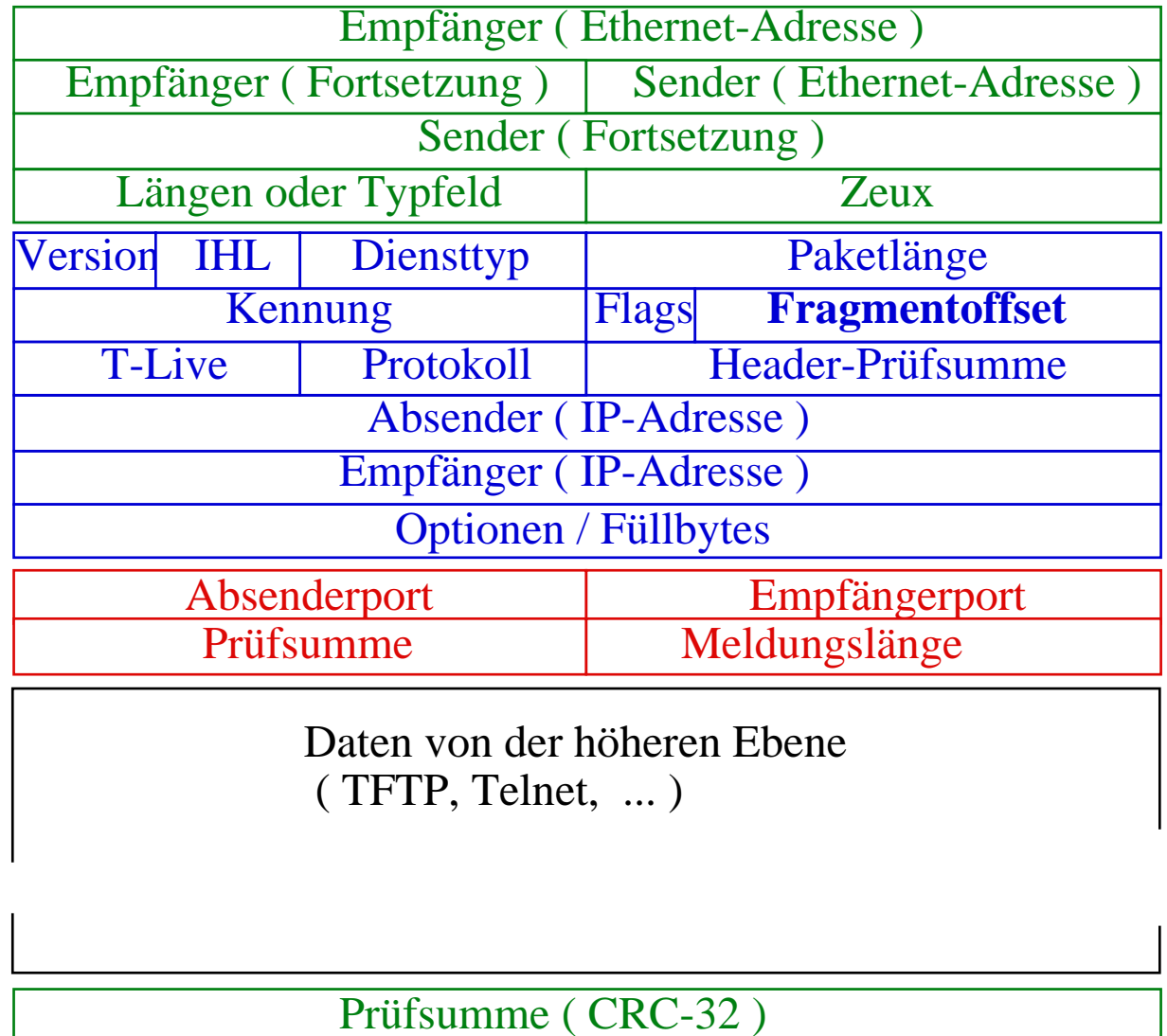
- **TCP Schicht:**

- Port-Nummern,
- Position im Bytestrom,
- Bestätigung ...



12.1.3 UDP/IP Paketformat:

- Streamposition aus TCP entfällt.
- UDP einfacher als TCP:
 - oft für Medienströme,
 - keine Bestätigung,
 - IP Fragmente.
- Für LLC / IP siehe oben.
- **UDP Schicht:**
 - Viele TCP Felder entfallen
 - Portg Adressierung bleibt.



12.2. URL Adressierung

12.2.1 URL: Universal Resource Locator:

- Symbolische Namen für irgendein Objekt im Netz:
 - Protokoll, Port, Prozess, Rechner, Datei, Verzeichnis ... :
 - `ftp://enterpr.informatik.uni-ulm.de:21/Pub`
 - `telnet://voyager:80/`
- Numerisch als IP-Adresse, z.B. 134.60.77.73:
 - 134.60 für das Netz der Universität Ulm,
 - 77 für das Subnetz "Verteilte Systeme",
 - 73 für den Rechner "Enterprise".
- Evtl. mit Port & Dateinamen:
 - 134.60.77.74:`80/index.html` (Port 80 für Web-Service).
- Default Ports für verschiedene Protokolle:
 - http: 80, ftp: 21, telnet: 23 ...
- Evtl. mit Telnet an irgendeinem Port testen (nicht Port# 23).
- Relative URLs für Elemente von Web-Seiten.

12.2.2 DNS - Domain Name Service

- Gestattet die Umwandlung symbolischer Namen in ein IP-Adresse:
severin.rz.uni-ulm.de => 134.60.1.111
- URL-Extensions ursprünglich als Länderkennung, heute überregional:
 - de, com, edu, org, fr, uk, jp, dk, ch, it ...
- Namensraum wird in Domains eingeteilt, die eine Hierarchie auffalten.
- Verwaltung des Namensraumes:
 - durch eine lokale Datei "LMHOSTS",
 - durch Anfragen bei Name Servern (DNS),
 - rekursive Anfragen in einer NS-Hierarchie,
 - Caching von alten Anfragen im Server.
- Domain Präfix für abgekürzte Namen:
 - *severin* (*.rz.uni-ulm.de*),
 - *happy* (*.informatik.uni-ulm.de*),
 - werden normalerweise manuell eingetragen und durch einen Administrator vergeben.
- Automatische Vergabe von IP-Adressen und Konfigurierung durch DHCP-Server (Dynamic Host Configuration Program).

12.2.3 IP-Adressen:

- Geographisch benachbarte Rechner tragen ähnliche Adressen. Damit wird die Weglenkung für die Datenpakete erleichtert.
- Class A Adressen:

1 – 126	Host Address Part
---------	-------------------
- Class B Adressen:

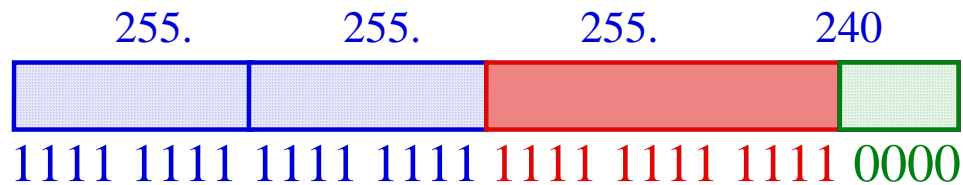
128.1 – 191.254	Host Address Part
-----------------	-------------------
- Class C Adressen:

192.0.1 – 223.255.254	Host
-----------------------	------
- Class D Adressen:

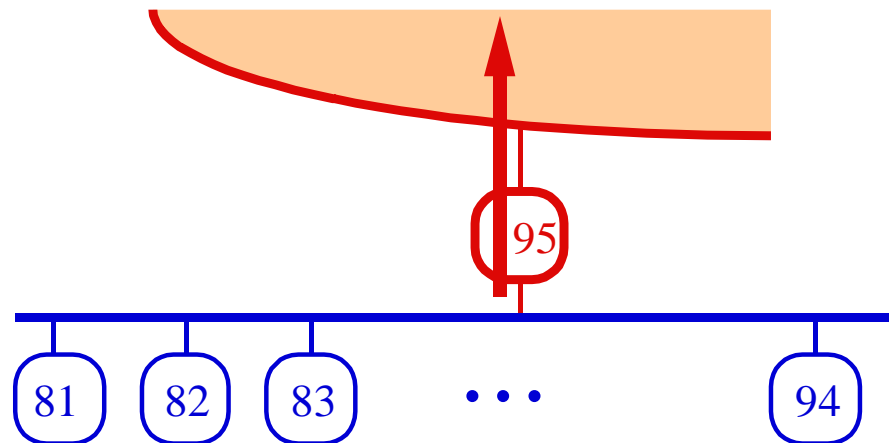
1110	Multicast - Kennung
------	---------------------
- Class E Adressen:

1111	Broadcast im Segment
------	----------------------
- Substruktur bei Class B & C Adressen.
- Substruktur entspricht nicht immer den physikalischen Netzsegmenten.

12.2.4 Subnetzadressmaske:



- Bestimmt, wann Router für Weiterleitung v. Paketen angesprochen wird.
- Hier Subnetz mit maximal 14 Stationen (z.B. 89.9.1.81 bis ...94):
 - Rundspruch über Adresse **base+0**, Default-Gateway über Adresse **base+15** ...
- Die Kommunikation zwischen 80..95 läuft direkt auf dem lokalen Netz.
- Pakete zu anderen Stationen laufen über Router (fälschlich "Gateway"):

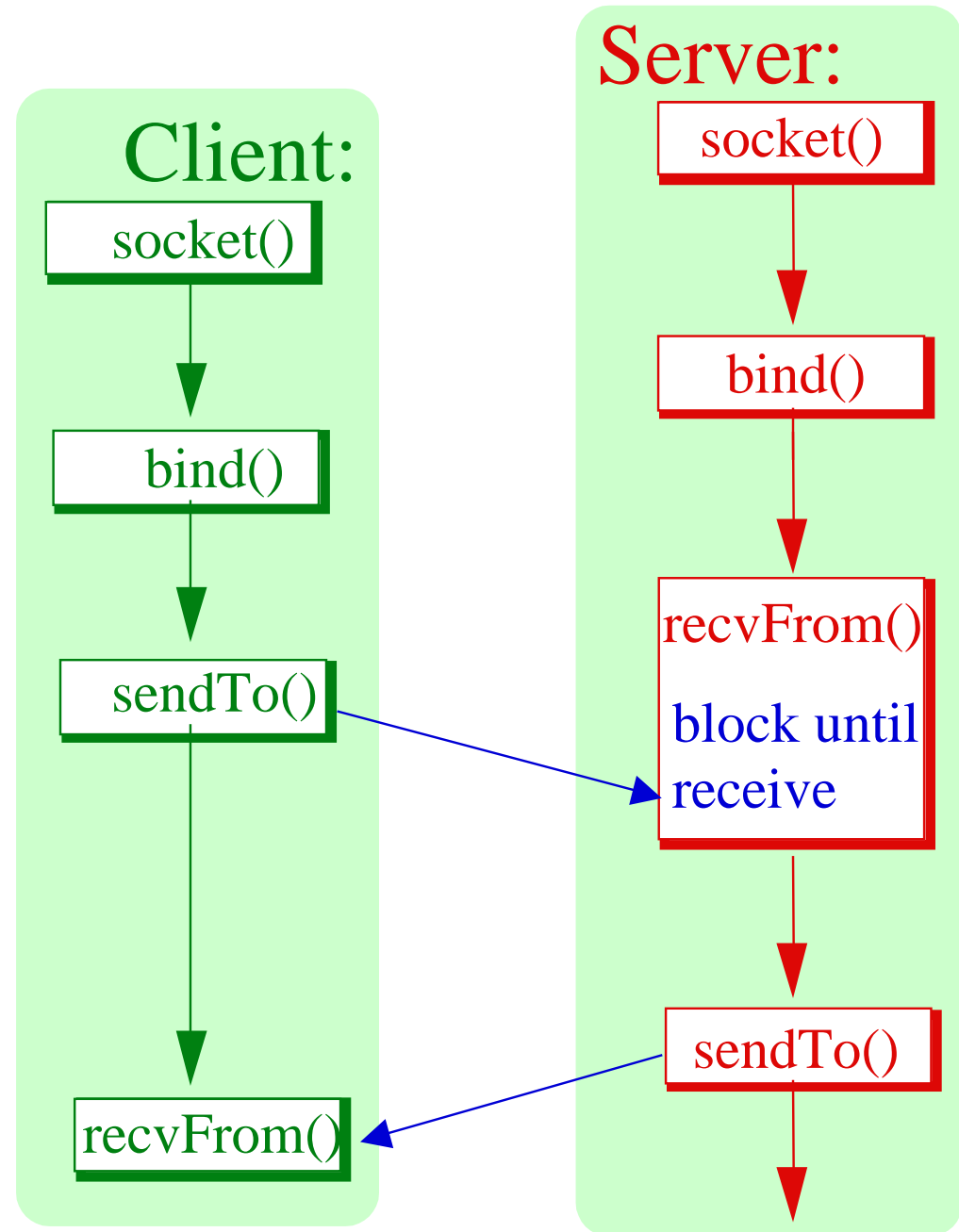


12.3. Winsock Sockets

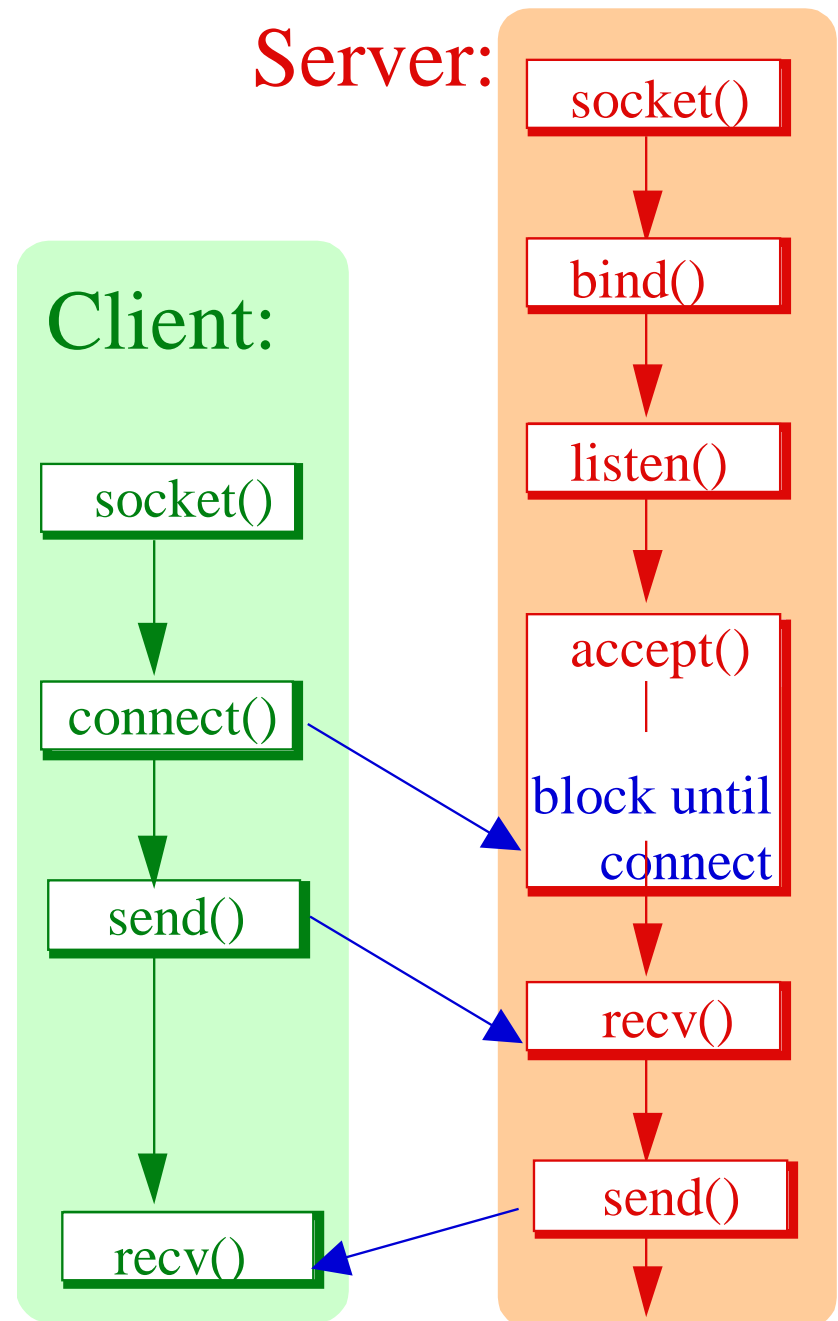
- Kommunikationsendpunkte.
- Werden an einen Port gebunden, der für einen bestimmten Dienst steht.
- API-Methoden (z.B. WinSock):
 - socket: erzeugt neuen Komm.-Endpunkt.
 - bind: **Port-Nummer im Socket eintragen.**
 - listen: Server Status etablieren (Q-Len).
 - accept: wartet auf einen "Anrufer" (Client)
 - connect: verbindet mit fremdem Port.
 - shutdown: Deaktivieren.
 - recv: **empfangen aus Verbindung.**
 - send: **senden in Verbindung.**
 - recvFrom: **verbindungslos empfangen.**
 - sendTo: **verbindungslos senden.**



12.3.1 Winsock verbindungslos:



12.3.2 Winsock verbindungsorientiert:



12.4. DatagramSocket Klasse in Java:

- Behandelt Pakete einzeln, nicht als Stream.
- Hat lokale IP-Adresse & einen Port.
- Oft keine Sequenzgarantie.
- Keine Quittung.

12.4.1 Beispielprogramm:

- Empfänger empfängt 5 Pakete.
- Senderklasse sendet 10 Pakete.
- **Textuelle Programmausgabe =>**
- Evtl. verzögerte Anzeige.

```
main end
sent 1
1 Send time was >20/01/04 20:25:57
2 Send time was >20/01/04 20:26:19
sent 2
sent 3
3 Send time was >20/01/04 20:26:20
sent 4
4 Send time was >20/01/04 20:26:21
sent 5
5 Send time was >20/01/04 20:26:22
empfaenger end
sent 6
sent 7
sent 8
sent 9
sent 10
sender end
```

12.4.2 Program "DGramExch" Teil 1 - Basisklasse:

- "Initialisierung":
 - Standard Packages importieren,
 - Puffer, Socket & Paket deklarieren,
 - Ferne IP-Adresse wählen,
 - Lokalen Port wählen.
- DatagramPacket:
 - mit `byte[]` Paketpuffer,
 - mit Zieladresse (IP-Adresse),
 - lokaler Port ...
- Zuerst Empfänger-Thread und dann Sender-Thread starten.

```
import java.io.*;
import java.net.*;
import java.util.*;

public class DGramExch extends Thread
{
    String msg;
    int packCount=0;
    DatagramSocket sock;
    DatagramPacket pack;
    byte[] buff=new byte[99];
    InetAddress ipAddr;
    static final int myPort=4711;
    static final String ipName
        =new String("127.0.0.1");

    public static void main(String[]s)
    {
        Epfngnr epf=new Epfngnr();
        Sender snd=new Sender();
        epf.start(); snd.start();
        System.out.println("main end");
    }
}
```

12.4.3 "DGramExch" Teil 2 - Empfänger:

- Socket-Konstruktor weist den Port zu (Bind entfällt).
- String-Konstruktor konvertiert Bytes in Unicode.
- Puffer wird vom Paket referenziert (Zeiger ?).
- Etwas gezwungene Ausnahmebehandlung.

```
class Epfngnr extends DGramExch
{
    public void run(){
        try
        {
            sock=new DatagramSocket(myPort);
            pack=new DatagramPacket(buff,99);
            while ( packCount++ <5 ) {
                sock.receive(pack);
                msg= new String(buff,0);
                System.out.println(msg.trim());
            };
        }
        catch (IOException ex){};
        sock.close();
        System.out.println("empfaenger end");
    }
}
```

12.4.4 "DGramExch" Teil 3 - Sender:

- Ratensteuerung über Wartezeiten (sleep()).

```
class Sender extends DGramExch
{
    public void run()
    {
        while (packCount++ <10)
        {
            try { sleep(999);}
            catch ( InterruptedException ix){}
            sendOne( packCount );
        };
        sock.close(); System.out.println("sender end");
    }
    void sendOne(int outPacks)
    {
        int len; Date now = new Date();
        msg=outPacks+" Send time is>"+ now.toLocaleString();
        len=msg.length();
        msg.getBytes(0,len,buff,0);
        try
        {
            sock=new DatagramSocket(myPort+1);
            ipAddr=InetAddress.getByName(ipName);
            pack=new DatagramPacket(buff,len,ipAddr,myPort);
            sock.send(pack);
            System.out.println("sent "+outPacks);
        }
        catch (IOException ex)    {}
    }
}
```

12.5. ServerSocket in Java

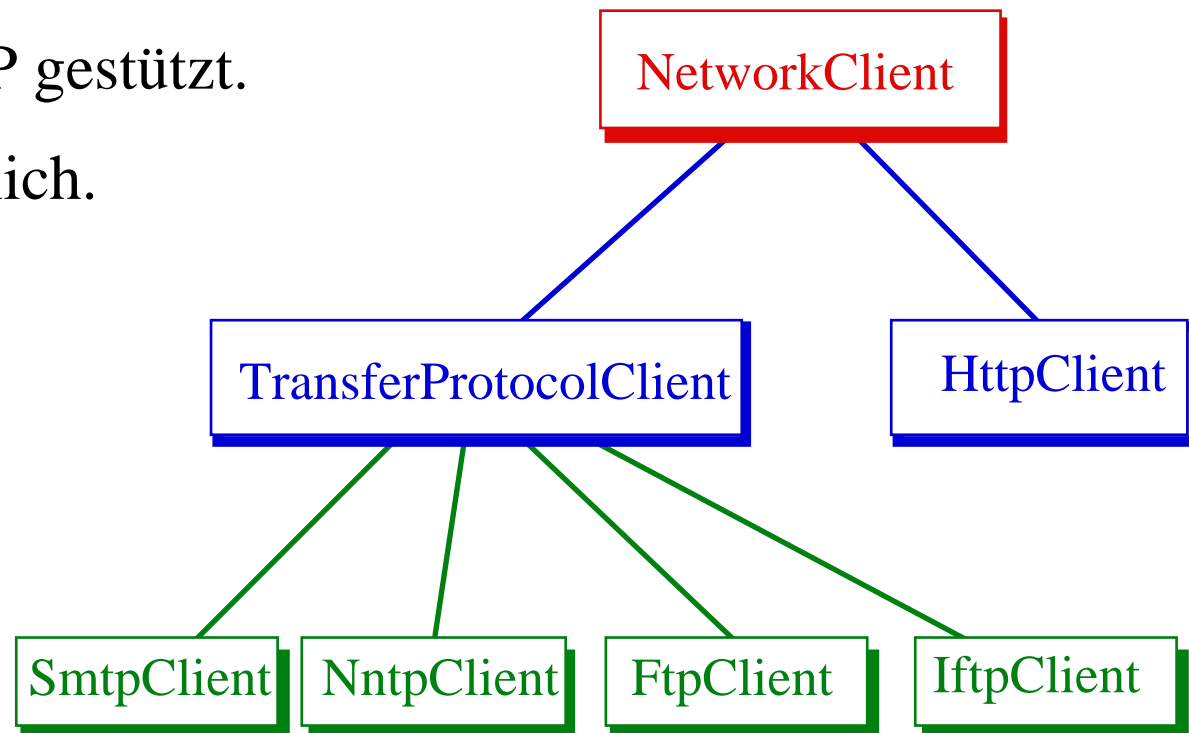
- Realisiert einen Server.
- *ServerSocket* horcht auf einem bestimmten Port *sock.listen()*.
- Thread blockiert in *sock.accept()* bis sich ein Klient anmeldet.
- Erst dann entsteht eine Connection zwischen zwei Ports.
- Der Server kann nun sein eigenes Protokoll abwickeln.
- Evtl. multithreaded für mehrere Klienten.
- Vorgefertigte Protokollhandler sind meist nur für Klientenstationen vorhanden (siehe *sun.net...*):
 - smtp: `java.sun.net.smtp.smtpClient.class`
 - nntp: `java.sun.net.nntp.nntpClient.class`
 - ftp: `java.sun.net.ftp.ftpClient.class`

12.5.1 Programm "EchoServerDemo"

```
import java.io.*;
import java.net.*;
public class EchoServerDemo
{
    public static void main(String[] s)
    {
        String msg;
        Socket conn;
        PrintStream out;
        ServerSocket sock;
        DataInputStream in;
        Try
        {
            sock = new ServerSocket(8189);
            conn = sock.accept();
            in = new DataInputStream(conn.getInputStream() );
            out = new PrintStream(conn.getOutputStream() );
            out.println("Hello, enter bye to exit.\r");
            do
            {
                msg= in.readLine();
                out.println("echo: "+msg+"\r");
                if (msg==null) break;
                if (msg.trim().equals("bye") ) break;
            } while(true);
            conn.close();
        }
        catch (IOException ioX){System.out.println(ioX)};
    }
}
```

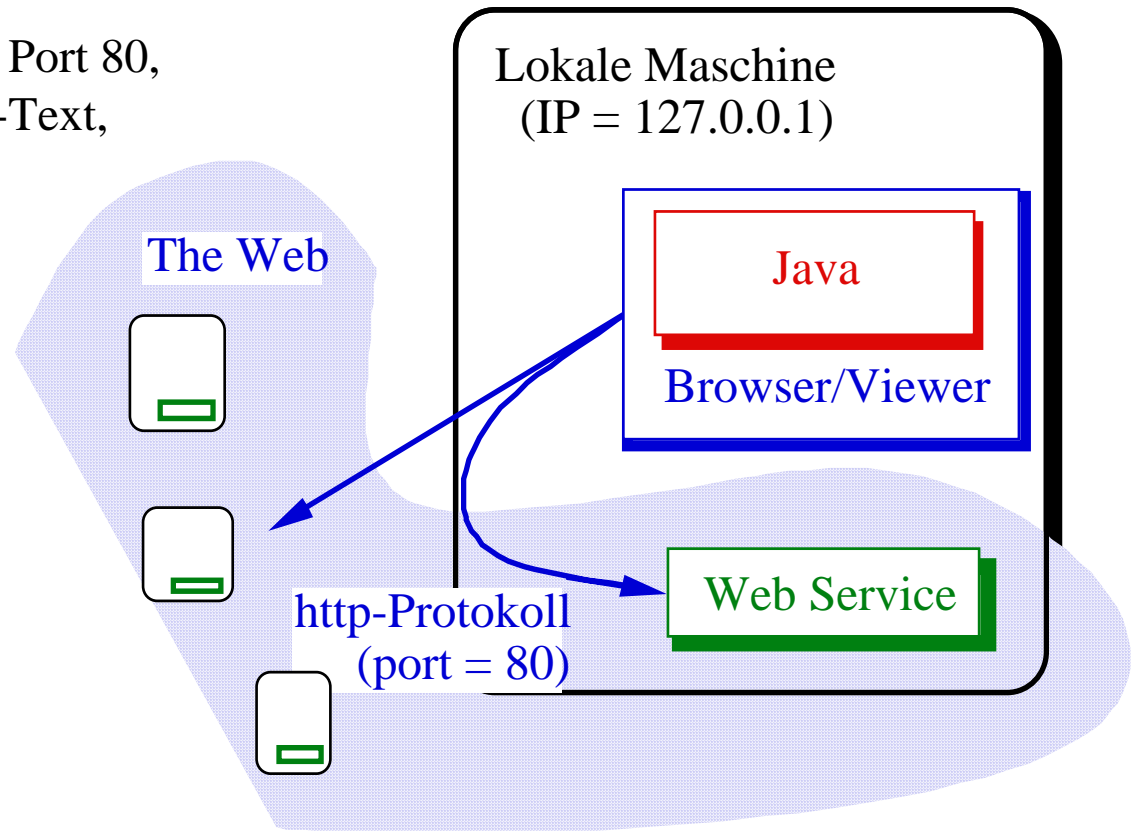
12.5.2 Protokollstammbaum in Java

- Zu finden unter java.sun.net. - Nicht in MS-java++
- Weitere Protokolle & Server von Netscape.
- Handler für Dateiformate (MIME).
- Ausschliesslich TCP/IP gestützt.
- Insgesamt unübersichtlich.



12.6. WWW-Szenarium:

- Zugriff über das Netz auf HTML-Seiten.
- Darstellung der Seiten in einem Browser.
- Hypertext-Links zur Navigation.
- World-Wide-Web Dienst:
 - WWW-Prozess normalerweise über Port 80,
 - Übertragung als formatierter ASCII-Text,
 - Zusätzlich Bilder, Ton und Applets,
 - LocalHost = 127.0.0.1 ...



12.6.1 Typische HTML-Seite:

- Formatierung mit Tags:

<XX> ... </XX> ...

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<head>
<title>Die Technik dahinter</title>
<meta name="GENERATOR" content="MSFrontPg1.1">
</head>
<body>
<p>¶;</p>
<h1 align=center>Die Technik hinter der Vernetzung
</h1>
<hr>
<ul>
<li><font size=6>PC Technik</font></li>
<li><a href="file://c|/pswitchg.gif">
<font size=6>Paketvermittlung</font></a></li>
<li><font size=6>Weltweite Vernetzung</font></li>
<li><font size=6>ISDN und schnelle
Netze</font></li>
<li><font size=6>mit geringen Kosten</font></li>
</ul>
</hr>
<h1>
<a href="vortrag.htm">Übersicht</a>
<a href="Chancen.htm">Weiter</a>
<a href="PersKomm.htm">Zurück</a>
</h1>
</body>
</html>
```

12.6.2 Beispielprogramm "URLStream":

- Liest eine Seite vom Port 80 eines Web-Servers.
- Formatierung entfällt, da hier kein Browser.
- HTTP impliziert per Default Port 80.
- Automatische Interpretation auch für FTP, TELNET, GOPHER, ECHO, NEWS, ...

```
import java.io.*;
import java.net.*;
public class URLStream
{   public static void main(String[]s)
    {   URL url;
        String zeile;
        URLConnection urlStream;
        InputStream iStream;
        DataInputStream diStream;
        try
        {   url=new URL("http://127.0.0.1/abc.htm");
            urlStream=url.openConnection();
            iStream=urlStream.getInputStream();
            diStream=new DataInputStream(iStream);
            while (true)
            {   zeile=diStream.readLine();
                if (zeile==null) break;
                System.out.println(zeile);
            }
        }
        catch(IOException iX) {
            System.out.println(iX.toString());
        }
    }
};
```

12.7. Protokolle der Internet Suite

12.7.1 Internet Managementprotokolle

ARP:	Adress resolution protocol
RARP:	Reverse address resolution protocol
DNS:	Domain Name Service
ICMP:	Internet control message protocol
IGMP:	Internet group management protocol
SNMP:	Simple network management protocol
RIP:	Routing information protocol

12.7.2 Anwendungsprotokolle

FTP:	File transfer protocol
SMTP:	Simple mail transfer protocol
NFS:	Network file system
NNTP:	News network transfer protocol
Telnet:	Virtuelles Netzwerk Terminal.
rLogin:	Remote Login auf fremder Station
HTTP:	Hyper-text transfer protocol
Archie:	Datenbanksystem über FTP-Server
Gopher:	Allg. verteiltes Informationssystem

12.8. Wellknown Ports

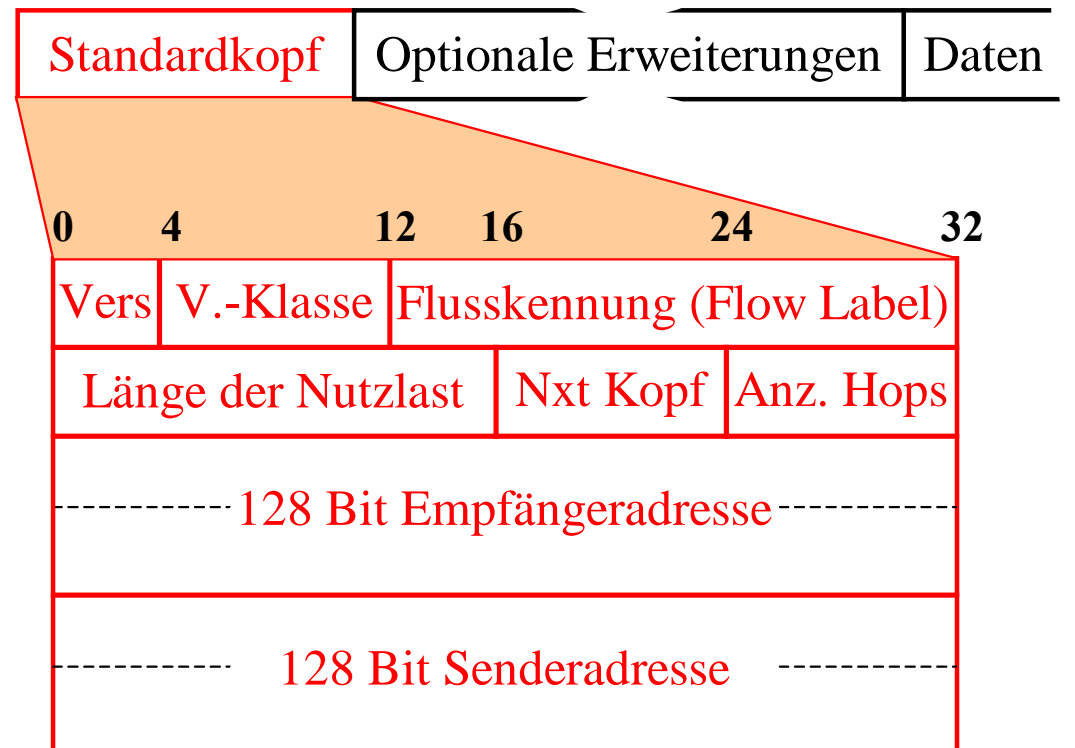
- Einige der 65535 Ports sind fest zugeordnet (für TCP & UDP getrennt):

Port #	Protokoll	
21	FTP	Dateiübertragung vom Server (TCP basiert)
23	Telnet	Terminalbetrieb als Konsole
25	SMTP	Einfacher Postübertragungsprotokoll
53	DNS	Netzweiter hierarchischer Namensdienst
67	Bootstrap Service	Starten über das Netz, IP# Vergabe ...
68	Bootstrap Klient	Starten, IP# Vergabe, Boot-Datei ...
69	Trivial FTP	Dateiübertragung auf UDP-Basis
79	Finger	Bericht über ferne User & Programme
80	HTTP	WWW-Seiten für Browser
88	Kerberos Authentifizierung	
109	POP2	Post-Office Protocol 2
110	POP3	Post-Office Protocol 3
111	Sun RPC	Prozeduraufruf aus der Ferne
161	SNMP	Einfacher Netzmanagementprotokoll
513	rLogin	Login aus der Ferne
520	RIP	Weglenkungsinformation
...	...	

12.9. Neues IP Protokoll (IPv6)

12.9.1 Motivierung

- Auf Grund des gewaltigen Wachstums sind Adressen knapp geworden.
- Januar 2003 ca. 170 Millionen Internet Hosts (registriert in DNS).
- Nicht alle 2^{32} Adressen können genutzt werden (Subnetzorganisation).
- IPv4 (IP-Version 4) ist das aktuelle Internetprotokoll:
 - weitgehend statische Adressenzuweisung,
 - ungenügender Adressbereich (32 Bit),
 - keine Durchsatzgarantien ...
- IPv6 als Weiterentwicklung von IPv4:
 - 128 Bit Adressen,
 - autom. Adresszuweisung,
 - Vereinfachung des Paketkopfes,
 - fakultative Paketkopferweiterungen.
- Migration von IPv4 nach IPv6:
 - IPv4 in IPv6-Adressraum "mappen",
 - IPv6 aus IPv4 Netz heraus ansprechen.



- erweitertes Dienstangebot:
 - Unterstützung mobiler Systeme,
 - Autorisierung und Verschlüsselung,
 - Synchronisierung von Datenströmen (MM),
 - Durchsatzgarantien & Dienstqualität möglich ...

12.9.2 IPv6 Paketformat:

- Standard Paketkopf evtl. mit Erweiterungen:
 - Versionsnummer - IPv4 oder IPv6,
 - Verkehrsklasse z.B. als Dienstqualität,
 - Flowlabel z.B. als Weg/Röhre durch das Netz,
 - Nutzlast maximal 65535 Bytes, keine Fragmente,
 - Übergabe an den nachfolgenden Protokollhandler,
 - maximal zulässige Teilstrecken, Hop Limit.
- Verschiedene Adressenformate:
 - Unicast-, Multicast-, Anycast-Adressen,
 - Aggregatable Global Unicast Address,
 - Link-local Address (in Subnetz),
 - Site-local Address.

12.10. Windows Sockets in C

- Winsock-Versionen 1.1 und 2.0:
 - Unter anderem QoS und Multicast in Winsock2.0,
 - http://www.stardust.com/winsock/ws_vers.htm.
- Etwas umständlicher als in Java:
 - Winsock-DLL initialisieren,
 - Byte-Reihenfolge berücksichtigen,
 - Adressinformation explizit zuweisen,
- Beispielprogramm für einen TCP-Klienten (Teil 1):

```

#include <stdio.h>           // formatted stream io etc.
#include <windows.h>        // special data types
#include <winsock.h>        // socket DLL for windows

int main( int argc, char **argv ) {
    WSADATA          wsa;           // version of winsock
    SOCKADDR_IN      destA;        // destination addr., internet type
    PHOSTENT          phe;         // host by name
    SOCKET            sock;        // socket structure
    char              buffer[256]; // packet buffer, 256 Bytes
    int               rslt;        // result of api-call

```

Beispielprogramm für einen TCP-Klienten (Teil 2):

```
rslt = WSStartup(MAKEWORD(1, 1), &wsa);           // initialize the Windows Socket DLL

phe = gethostbyname(argv[1]);                     // get Destin. IP-address via host name
memcpy( (char FAR *) & (destA.sin_addr), phe->h_addr, phe->h_length); // copy bytes
destA.sin_port = htons(10001);                   // Host to network order SHORT
destA.sin_family = AF_INET;                      // Internet address family

sock = socket( AF_INET, SOCK_STREAM, 0);          // create a TCP socket
rslt = connect( sock, (LPSOCKADDR) & destA, sizeof(destA) ); // connect to server
sprintf(buffer, "Hello Server");                 // build packet for the server
rslt = send(dSkt, buffer, strlen(buffer), 0);     //byteCnt, send message, no flags

closesocket(dSkt);                               // no more messages
WSACleanup();                                    // no more sockets
return 0;                                        // error codes ignored, beware !
}
```

12.10.1 Ausschnitt aus einem TCP-Server:

- Für ServerSockets expliziter *bind()* erforderlich.
- *bind()* bestimmt den Port, an welchem der Socket hören soll,
- *listen()* schaltet den ServerSocket ein,
- *accept()* liefert einen Socket für jede ankommende Verbindung.

```
serverSockAddr.sin_addr.s_addr    = htonl( INADDR_ANY);    // host to network long
serverSockAddr.sin_family         = AF_INET;                // Internet address family
serverSockAddr.sin_port           = htons(10002);           // host to network short
status = bind( serverSocket      // associates socket with address
              , (LPSOCKADDR) &serverSockAddr, sizeof(serverSockAddr) );

status=listen( serverSocket, 1);    // allow the socket to take connections

clientSocket = accept( serverSocket, // accept connect request, block until one is received
                      , (LPSOCKADDR) &clientSockAddr, &addrLen );

numrcv = recv( clientSocket        // receive data
              , buffer, MAXBUFLen-1, NO_FLAGS_SET);
```

12.10.2 API für WinSock-Socketfunktionen

Namensdienstfunktionen:

<code>gethostbyaddr</code>	Liefert den Rechnernamen zur IP-Adresse
<code>gethostbyname</code>	Liefert Rechner-Adresse über den Rechnernamen
<code>gethostname</code>	Liefert Name des lokalen Rechners
<code>getprotobyname</code>	Protokoll über den bekannten Protokollnamen
<code>getprotobynumber</code>	Protokoll über die Protokollnummer
<code>getservbyname</code>	Serverport zu einem Dienstnamen
<code>getservbyport</code>	Dienst-Information über die Portnummer

Basisfunktionen für Winsock (TCP & UDP):

<code>accept</code>	Akzeptiert eine Verbindung mit einem wartenden Socket und gibt neu erzeugten, verbundenen Socket zurück.
<code>bind</code>	Bindet lokale Adresse und Port an einen Socket.
<code>closesocket</code>	Schließt einen Socket und gibt einen Deskriptor frei.
<code>connect</code>	Richtet eine Verbindung zu einem Remote-Rechner ein.
<code>getpeername</code>	Holt Adresse und Port des Remote-Rechners.

getsockname	Holt Adresse und Port eines Sockets.
getsockopt	Holt den Wert einer Option eines Sockets.
ioctlsocket	Holt oder setzt die Parameter eines Sockets.
listen	Lässt Socket auf ankommende Verbindungen warten.
recv	Empfängt von einem Socket.
recvfrom	Empfängt vom Socket und gibt RemoteAdresse zurück.
select	Warten auf Lese-/Schreibbereitschaft mehrerer Sockets.
send	Sendet Daten zu einem verbundenen Socket.
sendto	Sendet zu expliziter Remote-Rechneradresse und -Port.
setsockopt	Setzt eine Option für einen lokalen Socket.
shutdown	Deaktiviert Senden und Empfangen auf einem Socket.
socket	Erzeugt einen Socket.